

BBBBBBBB	000000	000000	EEEEEEEEE	MM	MM	UU	UU	LL	AAAAAA	TTTTTTTTT
BBBBBBBB	000000	000000	EEEEEEEEE	MM	MM	UU	UU	LL	AA	TT
BB	BB	00	00	00	00	EE	MM	MM	AA	TT
BB	BB	00	00	00	00	EE	MM	MM	AA	TT
BB	BB	00	00	00	00	EE	MM	MM	AA	TT
BB	BB	00	00	00	00	EE	MM	MM	AA	TT
BBBBBBBB	00	00	00	00	00	EEEEE	MM	MM	AA	TT
BBBBBBBB	00	00	00	00	00	EEEEE	MM	MM	AA	TT
BB	BB	00	00	00	00	EE	MM	MM	AAAAAAA	TT
BB	BB	00	00	00	00	EE	MM	MM	AAAAAAA	TT
BB	BB	00	00	00	00	EE	MM	MM	AA	TT
BB	BB	00	00	00	00	EE	MM	MM	AA	TT
BBBBBBBB	000000	000000	EEEEEEEEE	MM	MM	UUUUUUUUU	LLLLLLL	AA	AA	TT
BBBBBBBB	000000	000000	EEEEEEEEE	MM	MM	UUUUUUUUU	LLLLLLL	AA	AA	TT

LL		SSSSSSS
LL		SSSSSSS
LL		SS
LL		SS
LL		SS
LL		SSSSS
LL		SSSSS
LL		SS
LL		SS
LL		SS
LL		SSSSSSS
LL		SSSSSSS

(2)	119	DECLARATIONS
(3)	188	VAX\$EMULATE - Entry Path into Emulator
(4)	324	VAX\$EMULATE_FPD - Alternate Entry Path into Emulator
(5)	438	Dispatch Tables
(6)	588	Description of instruction-specific routines
(9)	749	CMPC3 - Exception handler for CMPC3 instruction
(10)	794	CMPC5 - Exception handler for CMPC5 instruction
(13)	937	LOCC - Exception handler for LOCC instruction
(34)	1953	Common Exit Path for VAX\$xxxxxx Routines

BOOSEMULATE
V04-000

K 10
- Subset VAX-11 Instruction Emulator for 16-SEP-1984 01:37:09 VAX/VMS Macro V04-00
19-MAY-1983 17:28:36 [EMULAT.SRC]BOOTSWT.MAR;1 Page 1
(1)

00000001 0000 1 BOOT_SWITCH = 1 ; Include bootstrap emulation subset

BOO
V04

```
0000 1 .NOSHOW CONDITIONALS
0000 2 .TITLE BOOSEMULATE - Subset VAX-11 Instruction Emulator for VMB
0000 3 .IDENT /V04-000/
0000 4 :
0000 5 ****
0000 6 :
0000 7 :* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 :* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 :* ALL RIGHTS RESERVED.
0000 10 :* ****
0000 11 :* ****
0000 12 :* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 13 :* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 14 :* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 15 :* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 16 :* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 17 :* TRANSFERRED.
0000 18 :* ****
0000 19 :* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 20 :* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 21 :* CORPORATION.
0000 22 :* ****
0000 23 :* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 24 :* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 25 :* ****
0000 26 :* ****
0000 27 :* ****
0000 28 :* ****
0000 29 :* ****
0000 30 :* ****
0000 31 :* ****
0000 32 :*
0000 33 :*
0000 34 :+++
0000 35 : Facility:
0000 36 :     VAX-11 Instruction Emulator
0000 37 : Abstract:
0000 38 :
0000 39 :     This is the main body of the instruction emulator that supports
0000 40 :     the instructions that are not a part of the microVAX architecture.
0000 41 :     The current design calls for support of the string instructions
0000 42 :     (including CRC), the decimal instructions, and EDITPC.
0000 43 :
0000 44 :     This routine performs the following steps.
0000 45 :
0000 46 :     o Moves operands from the exception stack to registers in an
0000 47 :         instruction-specific manner
0000 48 :     o Calls an instruction-specific subroutine to do the actual work
0000 49 :
0000 50 :     If errors occur along the way, those errors are reflected to the
0000 51 :     user as exceptions.
0000 52 :
0000 53 : Environment:
0000 54 :
0000 55 :     These routines run at any access mode, at any IPL, and are AST
0000 56 :     reentrant. The routine starts execution in the access mode and
0000 57 :     at the IPL at which the instruction executed.
0000 58 :
0000 59 :
0000 60 :
0000 61 :
```

0000 62 : Author:
0000 63 :
0000 64 : Lawrence J. Kenah
0000 65 :
0000 66 : Creation Date
0000 67 :
0000 68 : 17 August 1982
0000 69 :
0000 70 : Modified by:
0000 71 :
0000 72 : V01-011 LJK0041 Lawrence J. Kenah 16-Jul-1984
0000 73 : Clear FPD in saved PSL at VAX\$EMULATE_FPD entry so that
0000 74 : next instruction can execute correctly.
0000 75 :
0000 76 : V01-010 LJK0031 Lawrence J. Kenah 5-Jul-1984
0000 77 : Set R2 and R4 unconditionally to zero in EDITPC routine
0000 78 : to allow the storage of FPD flags and similar data.
0000 79 :
0000 80 : V01-009 LJK0026 Lawrence J. Kenah 19-Mar-1984
0000 81 : Perform final cleanup pass. Eliminate xxx_UNPACK routine
0000 82 : references. Add C-bit optimization to MOVP.
0000 83 :
0000 84 : V01-008 LJK0010 Lawrence J. Kenah 8-Nov-1983
0000 85 : Eliminate code in EXIT_EMULATOR path that unconditionally
0000 86 : clears the T-bit and conditionally sets the TP-bit. The
0000 87 : TP-bit is handled by the base hardware.
0000 88 :
0000 89 : V01-007 KDM0088 Kathleen D. Morse 20-Oct-1983
0000 90 : Make branches to VAX\$REFLECT_TO_VMS into jumps, so that
0000 91 : the bootstrap emulator will link without truncation errors
0000 92 : until that routine is finished.
0000 93 :
0000 94 : V01-006 KDM0003 Kathleen D. Morse 18-Apr-1983
0000 95 : Generate abbreviated VAX\$EMULATE_FPD for the bootstrap
0000 96 : emulator.
0000 97 :
0000 98 : V01-005 LJK0006 Lawrence J. Kenah 16-Mar-1983
0000 99 : Generate case tables with macros. Allow subset emulator
0000 100 : for bootstrap instruction emulation.
0000 101 :
0000 102 : V01-004 KDM0002 Kathleen D. Morse 16-Mar-1983
0000 103 : Fix fourth and fifth operand fetches for SUBP6, ADDP6,
0000 104 : MULP and DIVP.
0000 105 :
0000 106 : V01-003 KDM0001 Kathleen D. Morse 04-Mar-1983
0000 107 : Longword align the exception handler entry points.
0000 108 :
0000 109 : V01-002 LJK0005 Lawrence J. Kenah 15-Nov-1982
0000 110 : Use hardware aids provided by microVAX architecture revision.
0000 111 : Exception is now reported in caller's mode. Operands are parsed
0000 112 : and placed on the exception stack as exception parameters.
0000 113 :
0000 114 : V01-001 LJK0002 Lawrence J. Kenah 17-Aug-1982
0000 115 : Original version using kernel mode exception through OPCDEC
0000 116 : exception vector.
0000 117 :--

```
0000 119 .SUBTITLE DECLARATIONS
0000 120
0000 121 ; Include files:
0000 122
0000 123 .SOPDEF : Values for instruction opcodes
0000 124 $PSLDEF : Define bit fields in PSL
0000 125
0000 126 .NOCROSS : No cross reference for these
0000 127 .ENABLE SUPPRESSION : No symbol table entries either
0000 128
0000 129 .PACK_DEF : Stack usage when restarting instructions
0000 130 STACR_DEF : Stack usage for original exception
0000 131
0000 132 .DISABLE SUPPRESSION : Turn on symbol table again
0000 133 .CROSS : Cross reference is OK now
0000 134
0000 135 ; Macro definitions
0000 136
0000 137 .MACRO INIT_CASE_TABLE SIZE,BASE,ERROR_EXIT
0000 138 BASE:
0000 139 .REPT SIZE
0000 140 .WORD ERROR_EXIT-BASE
0000 141 .ENDR
0000 142 .ENDM INIT_CASE_TABLE
0000 143
0000 144 .MACRO CASE_TABLE_ENTRY OPCODE,-
0000 145 ROUTINE,-
0000 146 FPD_ROUTINE,-
0000 147 BOOT_FLAG
0000 148 SIGN_EXTEND OPS 'OPCODE ...OPCODE
0000 149 :..OFFSET = ...OPCODE - OPCODE_BASE
0000 150 .IF NOT DEFINED BOOT_SWITCH
0000 151 INCLUDE 'OPCODE = 0
0000 152 .EXTERNAL VAX$'OPCODE
0000 153 .EXTERNAL FPD_ROUTINE
0000 154 : = CASE_TABLE_BASE + <2 * ...OFFSET>
0000 155 .WORD ROUTINE - CASE_TABLE_BASE
0000 156 : = FPD_CASE_TABLE_BASE + <2 * ...OFFSET>
0000 157 .WORD -FPD_ROUTINE - FPD_CASE_TABLE_BASE
0000 158 .IF_FALSE
0000 159 IF IDENTICAL <BOOT_FLAG>,BOOT
0000 160 INCLUDE 'OPCODE = 0
0000 161 .EXTERNAL VAX$'OPCODE
0000 162 : = CASE_TABLE_BASE + <2 * ...OFFSET>
0000 163 .WORD ROUTINE - CASE_TABLE_BASE
0000 164 .ENDC
0000 165 .ENDC
0000 166 .ENDM CASE_TABLE_ENTRY
0000 167
0000 168 ; External declarations for exception handling
0000 169
0000 170 .DISABLE GLOBAL
0000 171
0000 172
0000 173 .EXTERNAL VAX$_OPCDEC, -
0000 174 VAX$_OPCDEC_FPD
0000 175
0000 176
0000 177
0000 178
0000 179
```

0000 180 : PSECT Declarations:
0000 181
0000 182 .DEFAULT DISPLACEMENT , WORD
0000 183
00000000 184 .PSECT _VAX\$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, QUAD
0000 185
0000 186 .NOSHOW CONDITIONALS

188 .SUBTITLE VAX\$EMULATE - Entry Path into Emulator
189 +
190 Functional Description:
191
192 There are two different entries into this module. When a reserved
193 instruction is first encountered, its operands are parsed by the
194 hardware (or microcode, if you will) and placed on the stack as
195 exception parameters. The code at address VAX\$EMULATE is then entered
196 through the ^XC8(SCB) exception vector. That routine dispatches to an
197 instruction-specific routine called VAX\$xxxxxx (xxxxxx represents the
198 name of the reserved instruction) after placing the operands into
199 registers as required by VAX\$xxxxxx.
200
201 If an exception occurred during instruction emulation such that a
202 reserved instruction executed again, this time with FPD set, then a
203 different exception path is taken. The stack has a different (smaller)
204 set of parameters for the FPD exception. A different
205 instruction-specific routine executes to unpack saved intermediate
206 state before resuming instruction emulation.
207
208 The access mode and IPL are preserved across either exception.
209
210 Input Parameters:
211
212 00(SP) - Opcode of reserved instruction
213 04(SP) - PC of reserved instruction (old PC)
214 08(SP) - First operand specifier
215 12(SP) - Second operand specifier
216 16(SP) - Third operand specifier
217 20(SP) - Fourth operand specifier
218 24(SP) - Fifth operand specifier
219 28(SP) - Sixth operand specifier
220 32(SP) - Seventh operand specifier (currently unused)
221 36(SP) - Eighth operand specifier (currently unused)
222 40(SP) - PC of instruction following reserved instruction (new PC)
223 44(SP) - PSL at time of exception
224
225 Notes on input parameters:
226
227 1. The information that appears on the stack for each operand depends
228 on the nature of the operand.
229
230 .rx - Operand value
231 .ax - Operand address
232 .wx - Operand address (Register destination is stored in one's
233 complement form. See VAX\$CVTPL for details.)
234
235 2. The old PC value is not used unless an exception such as an access
236 violation occurs and the instruction has to be backed up.
237
238 3. The seventh and eighth operands are not used for any existing VAX-11
239 instructions. Those slots in the exception stack frame are reserved
240 for future expansion.
241
242 4. The two PC parameters and the PSL are the only data that needs to
243 be preserved once the instruction-specific routine is entered.
244

```

0000 245 : Output Parameters:
0000 246
0000 247 : The operands are moved from the stack to general registers in a way
0000 248 : that varies from instruction to instruction. Control is transferred
0000 249 : to a specific routine for each opcode.
0000 250
0000 251 : Notes:
0000 252
0000 253 : There are several tables in the emulator that use the opcode as an
0000 254 : index. We choose to interpret the opcode as a signed quantity because
0000 255 : this reduces the amount of wasted space in the tables. In either case,
0000 256 : there are 27 useful entries.
0000 257
0000 258 : Unsigned opcode
0000 259
0000 260 : OPCODE_BASE = CVTPS (value of 8)
0000 261 : OPCODE_MAX = CVTLP (value of F9)
0000 262
0000 263 : TABLE_SIZE = 241 decimal bytes
0000 264
0000 265 : Signed opcode
0000 266
0000 267 : OPCODE_BASE = ASHP (value of F8 or -8)
0000 268 : OPCODE_MAX = SKPC (value of 3B)
0000 269
0000 270 : TABLE_SIZE = 67 decimal bytes
0000 271
0000 272 : The savings of more than 170 entries in each table justifies all
0000 273 : of the machinations that we go through to treat opcodes as signed
0000 274 : quantities.
0000 275 :-.
0000 276
0000 277 : Because the assembler does not understand sign extension of byte and
0000 278 : word quantities, we must accomplish this sign extension with macros. The
0000 279 : assignment statements that appear as comments illustrate the sense of the
0000 280 : macro invocations that immediately follow.
0000 281
0000 282 : OPCODE_MAX = OPS_SKPC ; Largest opcode in this emulator
0000 283
0000 284 : SIGN_EXTEND OPS_SKPC , OPCODE_MAX
0000 285
0000 286 : We further restrict the table size and supported operations when we are
0000 287 : building the bootstrap subset of the emulator. We only allow certain string
0000 288 : instructions to contribute to the emulator.
0000 289
0000 290 : OPCODE_BASE = OPS_CMPC3 ; Smallest (in signed sense) opcode
0000 291
0000 292 : SIGN_EXTEND OPS_CMPC3 , OPCODE_BASE
0000 293
0000 294 CASE_TABLE_SIZE = <OPCODE_MAX - OPCODE_BASE> + 1 ; Define table size
0000 295
0000 296 .ALIGN LONG ; Alignment for exception vector
0000 297
0000 298 VAX$EMULATE::
0000 299
0000 300
0000 301
0000 302
0000 303
0000 304
0000 305
0000 306 CASEB OPCODE(SP),#OPCODE_BASE,#<OPCODE_MAX-OPCODE_BASE>
0000 307

```

	0004	308	INIT_CASE_TABLE CASE_TABLE_SIZE,CASE_TABLE_BASE,10\$	
	002A	309		
	002A	310	: If we drop through the case dispatcher, then the fault was not caused	
	002A	311	: by executing one of the instructions supported by this emulator. Such	
	002A	312	: exceptions will simply be passed through to VMS. (In the bootstrap emulator,	
	002A	313	: there is no operating system to reflect the exception. We simply HALT.)	
00000000'8F	DD 0D	002A 0030	314 10\$: PUSHL #VAX\$_OPCDEC	: Store signal name
	DD	0032	315 PUSHL #13	; Total of 13 longwords in signal array
	00	0032	316 HALT	; Nothing else we can do

0033 326
0033 325
0033 326
0033 327
0033 328
0033 329
0033 330
0033 331
0033 332
0033 333
0033 334
0033 335
0033 336
0033 337
0033 338
0033 339
0033 340
0033 341
0033 342
0033 343
0033 344
0033 345
0033 346
0033 347
0033 348
0033 349
0033 350
0033 351
0033 352
0033 353
0033 354
0033 355
0033 356
0033 357
0033 358
0033 359
0033 360
0033 361
0033 362
0033 363
0033 364
0033 365
0033 366
0033 367
0033 368
0033 369
0033 370
0033 371
0033 372
0033 373
0033 374
0033 375
0033 376
0033 377
0033 378
0033 379
0033 380

.SUBTITLE VAXSEMULATE_FPD - Alternate Entry Path into Emulator

+ Functional Description:

This routine is entered through the ^XCC(SCB) exception vector when an instruction that is not a part of the microVAX architecture executes and the FPD bit is set in the PSL. The software state that was preserved by each instruction must be restored and instruction execution resumed. Access mode and IPL are preserved across the exception occurrence.

Before the various VAX\$xxxxxx (or VAX\$xxxxxx RESTART) routines regain control, this dispatcher must retrieve the delta PC from wherever it was stored and place the stack in the same state that it is in when the normal (FPD bit not set) instruction dispatcher passes control to the various VAX\$xxxxxx routines. The pictures below explain this.

Input Parameters:

- 00(SP) - PC of reserved instruction
- 04(SP) - PSL at time of exception

Output Parameters:

The following picture shows the state of the stack after the dispatcher has executed its preliminary code but before control is passed back to instruction-specific execution. Note that this routine makes the stack look like it does when a reserved instruction executes and FPD is not yet set. This is done to make the exception exit code independent of whether a different exception occurred while the emulator was running.

- 00(SP) - Return PC (Address of EXIT routine in this module)
- 04(SP) - Unused placeholder (OPCODE)
- 08(SP) - PC of reserved instruction (old PC)
- 12(SP) - Unused placeholder (OPERAND_1)
- 16(SP) - Unused placeholder (OPERAND_2)
- 20(SP) - Unused placeholder (OPERAND_3)
- 24(SP) - Unused placeholder (OPERAND_4)
- 28(SP) - Unused placeholder (OPERAND_5)
- 32(SP) - Unused placeholder (OPERAND_6)
- 36(SP) - Unused placeholder (OPERAND_7)
- 40(SP) - Unused placeholder (OPERAND_8)
- 44(SP) - PC of instruction following reserved instruction (new PC)
- 48(SP) - PSL at time of exception

Before this routine dispatches to opcode-specific code, it calculates the PC of the next instruction based on the PC of the reserved instruction and the delta-PC quantity that was stored as part of the instruction's intermediate state. Note that the delta PC quantity

$$\text{delta PC} = \text{new PC} - \text{old PC}$$

is stored in the upper bytes of one of the general registers, usually bits <31:24> of R0 or R2. The registers R0 through R3 are stored on the stack (in the space used for the first four operands when the

0033 381 : reserved instruction is first encountered) so that the same offsets
0034 382 : that were used to store the delta-PC can be used to retrieve it.
0035 383 :-
0036 384 .ALIGN LONG : Alignment for exception vector
0037 385 VAXSEMULATE_FPD:::
0038 386
0039 387
003A 388
003B 389
003C 390
003D 391
003E 392
003F 393
0040 394
0041 395
0042 396
0043 397
0044 398
0045 399
0046 400
0047 401
0048 402
0049 403
004A 404
004B 405
004C 406
004D 407
004E 408
004F 409
0050 410
0051 411
0052 412
0053 413
0054 414
0055 415
0056 416
0057 417
0058 418
0059 419
005A 420
005B 421
005C 422
005D 423
005E 424
005F 425
0060 426
0061 427
0062 428
0063 429
0064 430
0065 431
0066 432
0067 433 HALT : Nothing else we can do

00000000'8F DD 0034 429 PUSHL #VAXS_OPCDEC_FPD ; This is the signal name
04 DD 003A 430 PUSHL #4 ; Signal array has four longwords
00 003C 431
00 003D 432
00 003E 433

```

003D 438 .SUBTITLE Dispatch Tables
003D 439 :+ Functional Description:
003D 440 :+
003D 441 :+
003D 442 : The case tables for the two CASEB instructions are built with the
003D 443 : macros that are invoked here. Macros are used to guarantee that both
003D 444 : tables contain correct entries for a selected opcode at the same
003D 445 : offset.
003D 446 :
003D 447 : Assumptions:
003D 448 :
003D 449 : The CASE_TABLE_ENTRY macro assumes that the names of the respective
003D 450 : case tables are CASE_TABLE_BASE and FPD_CASE_TABLE_BASE.
003D 451 :
003D 452 : Notes:
003D 453 :
003D 454 : In the following lists, those FPD routines that do not have _FPD in
003D 455 : their names use the same JSB entry point for initial entry and after
003D 456 : restarting the instruction. In most of these cases, the register state
003D 457 : is the same for both starting and restarting. For the remaining cases,
003D 458 : there is not enough difference between the two cases to justify an
003D 459 : additional entry point. (See VAXSMOVTC for an example of this latter
003D 460 : situation.)
003D 461 :
003D 462 : The FPD routines that include _RESTART in their names have to do a
003D 463 : certain amount of work to restore the intermediate state from the
003D 464 : canonical registers before they can resume instruction execution.
003D 465 ;-
003D 466 :
003D 467 .SAVE : Remember current location counter
003D 468 :
003D 469 : First generate table entries for the string instructions
003D 470 :
003D 471 CASE_TABLE_ENTRY OPCODE=MOVTC,-
003D 472 ROUTINE=MOVTC,-
003D 473 FPD_ROUTINE=VAXSMOVTC
003D 474 :
003D 475 CASE_TABLE_ENTRY OPCODE=MOVTC,-
003D 476 ROUTINE=MOVTC,-
003D 477 FPD_ROUTINE=VAXSMOVTC
003D 478 :
003D 479 CASE_TABLE_ENTRY OPCODE=CMPC3,-
003D 480 ROUTINE=CMPC3,-
003D 481 FPD_ROUTINE=VAXSCMPC3,-
003D 482 BOOT_FLAG=BOOT
0006 483 :
0006 484 CASE_TABLE_ENTRY OPCODE=CMPC5,-
0006 485 ROUTINE=CMPC5,-
0006 486 FPD_ROUTINE=VAXSCMPC5,-
0006 487 BOOT_FLAG=BOOT
000E 488 :
000E 489 CASE_TABLE_ENTRY OPCODE=LOCC,-
000E 490 ROUTINE=LOCC,-
000E 491 FPD_ROUTINE=VAXSLOCC,-
000E 492 BOOT_FLAG=BOOT
0028 493 :
0028 494 CASE_TABLE_ENTRY OPCODE=SKPC,-

```

0028	495	ROUTINE=SKPC,- FPD_ROUTINE=VAX\$SKPC
0028	496	
0028	497	
0028	498 CASE_TABLE_ENTRY	OPCODE=SCANC,- ROUTINE=SCANC,- FPD_ROUTINE=VAX\$SCANC
0028	499	
0028	500	
0028	501	
0028	502 CASE_TABLE_ENTRY	OPCODE=SPANC,- ROUTINE=SPANC,- FPD_ROUTINE=VAX\$SPANC
0028	503	
0028	504	
0028	505	
0028	506 CASE_TABLE_ENTRY	OPCODE=MATCHC,- ROUTINE=MATCHC,- FPD_ROUTINE=VAX\$MATCHC
0028	507	
0028	508	
0028	509	
0028	510 CASE_TABLE_ENTRY	OPCODE=CRC,- ROUTINE=CR,C,- FPD_ROUTINE=VAX\$CRC
0028	511	
0028	512	
0028	513	
0028	514 : Now generate table entries for the decimal instructions	
0028	515	
0028	516 CASE_TABLE_ENTRY	OPCODE=ADDP4,- ROUTINE=ADDP4,- FPD_ROUTINE=VAX\$ADDP4
0028	517	
0028	518	
0028	519 CASE_TABLE_ENTRY	OPCODE=ADDP6,- ROUTINE=ADDP6,- FPD_ROUTINE=VAX\$ADDP6
0028	520	
0028	521	
0028	522	
0028	523 CASE_TABLE_ENTRY	OPCODE=ASHP,- ROUTINE=ASHP,- FPD_ROUTINE=VAX\$ASHP
0028	524	
0028	525	
0028	526	
0028	527	
0028	528 CASE_TABLE_ENTRY	OPCODE=CMPP3,- ROUTINE=CMPP3,- FPD_ROUTINE=VAX\$CMPP3
0028	529	
0028	530	
0028	531	
0028	532 CASE_TABLE_ENTRY	OPCODE=CMPP4,- ROUTINE=CMPP4,- FPD_ROUTINE=VAX\$CMPP4
0028	533	
0028	534	
0028	535	
0028	536 CASE_TABLE_ENTRY	OPCODE=CVTLP,- ROUTINE=CVTLP,- FPD_ROUTINE=VAX\$CVTLP_RESTART
0028	537	
0028	538	
0028	539	
0028	540 CASE_TABLE_ENTRY	OPCODE=CVTPL,- ROUTINE=CVTPL,- FPD_ROUTINE=VAX\$CVTPL_RESTART
0028	541	
0028	542	
0028	543	
0028	544 CASE_TABLE_ENTRY	OPCODE=CVTPS,- ROUTINE=CVTPS,- FPD_ROUTINE=VAX\$CVTPS
0028	545	
0028	546	
0028	547	
0028	548 CASE_TABLE_ENTRY	OPCODE=CVTPT,- ROUTINE=CVTPT,- FPD_ROUTINE=VAX\$CVTPT_RESTART
0028	549	
0028	550	
0028	551	

0028	552	CASE_TABLE_ENTRY	OPCODE=CVTSP,- ROUTINE=CVTSP,- FPD_ROUTINE=VAX\$CVTSP
0028	553		
0028	554		
0028	555	CASE_TABLE_ENTRY	OPCODE=CVTTP,- ROUTINE=CVTTP,- FPD_ROUTINE=VAX\$CVTTP_RESTART
0028	556		
0028	557		
0028	558		
0028	559	CASE_TABLE_ENTRY	OPCODE=DIVP,- ROUTINE=DIVP,- FPD_ROUTINE=VAX\$DIVP
0028	560		
0028	561		
0028	562		
0028	563	CASE_TABLE_ENTRY	OPCODE=MOV P,- ROUTINE=MOV P,- FPD_ROUTINE=VAX\$MOV P
0028	564		
0028	565		
0028	566		
0028	567	CASE_TABLE_ENTRY	OPCODE=MULP,- ROUTINE=MUL P,- FPD_ROUTINE=VAX\$MULP
0028	568		
0028	569		
0028	570	CASE_TABLE_ENTRY	OPCODE=SUBP4,- ROUTINE=SUBP4,- FPD_ROUTINE=VAX\$SUBP4
0028	571		
0028	572		
0028	573	CASE_TABLE_ENTRY	OPCODE=SUBP6,- ROUTINE=SUBP6,- FPD_ROUTINE=VAX\$SUBP6
0028	574		
0028	575		
0028	576	CASE_TABLE_ENTRY	OPCODE=EDITPC,- ROUTINE=EDITPC,- FPD_ROUTINE=VAX\$EDITPC_RESTART
0028	577		
0028	578		
0028	579		
0028	580	: EDITPC always seems to find itself in last place	
0028	581		
0028	582	CASE_TABLE_ENTRY	OPCODE=EDITPC,- ROUTINE=EDITPC,- FPD_ROUTINE=VAX\$EDITPC_RESTART
0028	583		
0028	584		
0028	585	.RESTORE	: Reset current location counter
00000030	586		

003D 588 .SUBTITLE Description of instruction-specific routines
003D 589
003D 590 :++
003D 591 : The instruction-specific routines do similar things. Rather than clutter up
003D 592 : each routine with the same comments, we will describe the steps that each
003D 593 : routine takes in this section.
003D 594
003D 595 : The input parameters to each routine are identical.
003D 596
003D 597
003D 598
003D 599
003D 600 :
003D 601 : OPCODE(SP) - Opcode of reserved instruction
003D 602 : OLD PC(SP) - PC of reserved instruction
003D 603 : OPERAND_1(SP) - First operand specifier
003D 604 : OPERAND_2(SP) - Second operand specifier
003D 605 : OPERAND_3(SP) - Third operand specifier
003D 606 : OPERAND_4(SP) - Fourth operand specifier
003D 607 : OPERAND_5(SP) - Fifth operand specifier
003D 608 : OPERAND_6(SP) - Sixth operand specifier
003D 609 : OPERAND_7(SP) - Seventh operand specifier (currently unused)
003D 610 : OPERAND_8(SP) - Eight operand specifier (currently unused)
003D 611 : NEW PC(SP) - PC of instruction following reserved instruction
003D 612 : EXCEPTION_PSL(SP) - PSL at time of exception
003D 613
003D 614
003D 615
003D 616
003D 617
003D 618
003D 619 : Outline of execution:
003D 620
003D 621 : The operands are loaded into registers as required by the instruction
003D 622 : specific routines. Routine headers for each routine contain detailed
003D 623 : descriptions.
003D 624
003D 625 : A routine of the form VAX\$xxxxxx (where xxxxxx is the instruction
003D 626 : name) is called to perform the actual work indicated by each
003D 627 : instruction.
003D 628
003D 629 : Common exit code executes to allow the condition codes returned by the
003D 630 : VAX\$xxxxxx routines to be passed back to the code that generated the
003D 631 : original exception.
003D 632
003D 633 : Notes:
003D 634
003D 635 : The following routines are constructed to be reasonably fast. In
003D 636 : particular, each instruction has its own separate routine, even though
003D 637 : several instructions differ only in the instruction-specific routine
003D 638 : to which final control is passed. Rather than share this common code
003D 639 : at the expense of another dispatch on opcode, we choose to duplicate
003D 640 : the common code.
003D 641 :--
003D 642

```

003D 749      .SUBTITLE    CMPC3 - Exception handler for CMPC3 instruction
003D 750 :+
003D 751 : Input Parameters:
003D 752 :
003D 753 : OPCODE(SP)
003D 754 : OLD PC(SP)
003D 755 : OPERAND_1(SP) - len.rw
003D 756 : OPERAND_2(SP) - src1addr.ab
003D 757 : OPERAND_3(SP) - src2addr.ab
003D 758 : OPERAND_4(SP)
003D 759 : OPERAND_5(SP)
003D 760 : OPERAND_6(SP)
003D 761 : OPERAND_7(SP)
003D 762 : OPERAND_8(SP)
003D 763 : NEW PC(SP)
003D 764 : EXCEPTION_PSL(SP)
003D 765 :
003D 766 : Output Parameters:
003D 767 :
003D 768 : R0<15:0> - len.rw
003D 769 : R1      - src1addr.ab
003D 770 : R3      - src2addr.ab
003D 771 :
003D 772 : Implicit Output:
003D 773 :
003D 774 : R0<31:16> - 0
003D 775 : R2      - UNPREDICTABLE
003D 776 :-
003D 777 :
003D 778 CMPC3:
003D 779 :
50 08 AE 3C 003D 780   MOVZWL OPERAND_1(SP),R0      ; R0<15:0> <- srclen.rw
51 0C AE D0 0041 781   MOVL    OPERAND_2(SP),R1      ; R1      <- src1addr.ab
53 10 AE D0 0045 782   MOVL    OPERAND_3(SP),R3      ; R3      <- src2addr.ab
0049 783 :
0049 784 : Now that the operands have been loaded, the only exception parameter
0049 785 : other than the PC/PSL pair that needs to be saved is the old PC. However,
0049 786 : there is no reason why the state of the stack needs to be altered and we
0049 787 : save two instructions if we leave the stack alone.
0049 788 :
0080'CF 9F 0049 789   PUSHAB  VAXSEXIT_EMULATOR    ; Store the return PC
FFB0' 31 004D 790   BRW     VAXSCMPC3          ; Do the actual work

```

0050 794 .SUBTITLE CMPC5 - Exception handler for CMPC5 instruction
 0050 795 :+
 0050 796 Input Parameters:
 0050 797
 0050 798 OPCODE(SP)
 0050 799 OLD PC(SP)
 0050 800 OPERAND_1(SP) - src1len.rw
 0050 801 OPERAND_2(SP) - src1addr.ab
 0050 802 OPERAND_3(SP) - fill.rb
 0050 803 OPERAND_4(SP) - src2len.rw
 0050 804 OPERAND_5(SP) - src2addr.ab
 0050 805 OPERAND_6(SP)
 0050 806 OPERAND_7(SP)
 0050 807 OPERAND_8(SP)
 0050 808 NEW PC(SP)
 0050 809 EXCEPTION_PSL(SP)
 0050 810
 0050 811 Output Parameters:
 0050 812
 0050 813 R0<15:0> - src1len.rw
 0050 814 R0<23:16> - fill.rb
 0050 815 R1 - src1addr.ab
 0050 816 R2<15:0> - src2len.rw
 0050 817 R3 - src2addr.ab
 0050 818
 0050 819 Implicit Output:
 0050 820
 0050 821 R0<31:24> - UNPREDICTABLE
 0050 822 R2<31:16> - 0
 0050 823 :-
 0050 824
 0050 825 CMPC5:
 0050 826
 50 10 AE 10 9C 0050 827 ROTL #16,OPERAND_3(SP),R0 : R0<23:16> <- fill.rb
 50 08 AE 80 0055 828 MOVW OPERAND_1(SP),R0 : R0<15:0> <- src1len.rw
 51 0C AE D0 0059 829 MOVL OPERAND_2(SP),R1 : R1 <- src1addr.ab
 52 14 AE 3C 005D 830 MOVZWL OPERAND_4(SP),R2 : R2<15:0> <- src2len.rw
 53 18 AE D0 0061 831 MOVL OPERAND_5(SP),R3 : R3 <sca- src2addr.ab
 0065 832
 0065 833 : Now that the operands have been loaded, the only exception parameter
 0065 834 : other than the PC/PSL pair that needs to be saved is the old PC. However,
 0065 835 : there is no reason why the state of the stack needs to be altered and we
 0065 836 : save two instructions if we leave the stack alone.
 0065 837
 0080'CF, FF94', 9F 0065 838 PUSHAB VAX\$EXIT EMULATOR : Store the return PC
 FF94', 31 0069 839 BRW VAX\$CMPC5 : Do the actual work

006C 937 .SUBTITLE LOCC - Exception handler for LOCC instruction
 006C 938 :+
 006C 939 : Input Parameters:
 006C 940 .
 006C 941 OPCODE(SP)
 006C 942 OLD PC(SP)
 006C 943 OPERAND_1(SP) - char.rb
 006C 944 OPERAND_2(SP) - len.rw
 006C 945 OPERAND_3(SP) - addr.ab
 006C 946 OPERAND_4(SP)
 006C 947 OPERAND_5(SP)
 006C 948 OPERAND_6(SP)
 006C 949 OPERAND_7(SP)
 006C 950 OPERAND_8(SP)
 006C 951 NEW PC(SP)
 006C 952 EXCEPTION_PSL(SP)
 006C 953 .
 006C 954 Output Parameters:
 006C 955 .
 006C 956 R0<15:0> - len.rw
 006C 957 R0<23:16> - char.rb
 006C 958 R1 - addr.ab
 006C 959 .
 006C 960 Implicit Output:
 006C 961 .
 006C 962 R0<31:24> - UNPREDICTABLE
 006C 963 :-
 006C 964 .
 006C 965 LOCC:
 006C 966 .
 50 08 AE 10 9C 006C 967 ROTL #16,OPERAND_1(SP),R0 ; R0<23:16> <- char.ab
 50 0C AE 80 0071 968 MOVW OPERAND_2(SP),R0 ; R0<15:0> <- len.rw
 51 10 AE D0 0075 969 MOVL OPERAND_3(SP),R1 ; R1 <- addr.ab
 0079 970 .
 0079 971 : Now that the operands have been loaded, the only exception parameter
 0079 972 : other than the PC/PSL pair that needs to be saved is the old PC. However,
 0079 973 : there is no reason why the state of the stack needs to be altered and we
 0079 974 : save two instructions if we leave the stack alone.
 0079 975 .
 0080'CF FF80' 9F 0079 976 PUSHAB VAX\$EXIT_EMULATOR ; Store the return PC
 31 007D 977 BRW VAX\$LOCC ; Do the actual work

0080 1953 .SUBTITLE Common Exit Path for VAX\$xxxxxx Routines
 0080 1954 :+
 0080 1955 : Functional Description:
 0080 1956 :
 0080 1957 : This is the common exit path for all instruction-specific routines.
 0080 1958 : The condition codes returned by the VAX\$xxxxxx routine are stored in
 0080 1959 : the exception PSL and control is passed back to the instruction stream
 0080 1960 : that executed the reserved instruction.
 0080 1961 :
 0080 1962 : Input Parameters:
 0080 1963 :
 0080 1964 : PSL contains condition code settings from VAX\$xxxxxx routine.
 0080 1965 :
 0080 1966 : OPCODE(SP) - Opcode of reserved instruction
 0080 1967 : OLD PC(SP) - PC of reserved instruction
 0080 1968 : OPERAND_1(SP) - First operand specifier
 0080 1969 : OPERAND_2(SP) - Second operand specifier
 0080 1970 : OPERAND_3(SP) - Third operand specifier
 0080 1971 : OPERAND_4(SP) - Fourth operand specifier
 0080 1972 : OPERAND_5(SP) - Fifth operand specifier
 0080 1973 : OPERAND_6(SP) - Sixth operand specifier
 0080 1974 : OPERAND_7(SP) - Seventh operand specifier (currently unused)
 0080 1975 : OPERAND_8(SP) - Eight operand specifier (currently unused)
 0080 1976 : NEW PC(SP) - PC of instruction following reserved instruction
 0080 1977 : EXCEPTION_PSL(SP) - PSL at time of exception
 0080 1978 :
 0080 1979 : Implicit Input:
 0080 1980 :
 0080 1981 : General registers contain architecturally specified values according
 0080 1982 : to specific instruction that was emulated.
 0080 1983 :
 0080 1984 : Implicit Output:
 0080 1985 :
 0080 1986 : Control is passed to the location designated by "new PC" with the
 0080 1987 : condition codes as determined by VAX\$xxxxxx. The EXIT routine also
 0080 1988 : preserves general registers.
 0080 1989 :-
 0080 1990 :
 0080 1991 : VAX\$EXIT_EMULATOR::
 7E DC 0080 1992 : MOVPSL -(SP) ; Save the new PSL on the stack
 0082 1993 :
 0082 1994 : Note that the next instruction makes no assumptions about the condition
 0082 1995 : codes in the saved PSL.
 0082 1996 :
 04 00 8E F0 0082 1997 : INSV (SP)+,#0,#4,-
 2C AE 0086 1998 : EXCEPTION_PSL(SP) ; Replace saved condition codes
 SE 28 C0 0088 1999 : ADDL #NEW_PC,SP ; Adjust stack pointer (discard old PC)
 02 008B 2000 : REI ; Return
 008C 2001 :
 00BC 2002 : .END

...OFFSET	= 0000000F	OPS_CVTGF	= 000033FD
...OPCODE	= 00000038	OPS_CVTGH	= 000056FD
BOOT_SWITCH	= 00000001	OPS_CVTGL	= 00004AFD
CASE_TABLE_BASE	00000004 R 02	OPS_CVTGW	= 000049FD
CASE_TABLE_SIZE	= 00000013	OPS_CVTHB	= 000068FD
CMPC3	0000003D R 02	OPS_CVTHD	= 0000F7FD
CMPC5	00000050 R 02	OPS_CVTHF	= 0000F6FD
EXCEPTION_PSL	= 0000002C	OPS_CVTHG	= 000076FD
INCLUDE_CMPC3	= 00000000	OPS_CVTHL	= 00006AFD
INCLUDE_CMPC5	= 00000000	OPS_CVTHW	= 000069FD
INCLUDE_LOCC	= 00000000	OPS_CVTLD	= 0000006E
LOCC	0000006C R 02	OPS_CVTLF	= 0000004E
NEW_PC	= 00000028	OPS_CVTLG	= 00004EFD
OPS_ACBD	= 0000006F	OPS_CVTLH	= 00006EFD
OPS_ACBF	= 0000004F	OPS_CVTLP	= 000000F9
OPS_ACBG	= 00004FFD	OPS_CVTPL	= 00000036
OPS_ACBH	= 00006FFD	OPS_CVTPS	= 00000008
OPS_ADDD2	= 00000060	OPS_CVTPT	= 00000024
OPS_ADDD3	= 00000061	OPS_CVTRDL	= 0000006B
OPS_ADDF2	= 00000040	OPS_CVTRFL	= 0000004B
OPS_ADDF3	= 00000041	OPS_CVTRGL	= 00004BFD
OPS_ADDG2	= 000040FD	OPS_CVTRHL	= 000068FD
OPS_ADDG3	= 000041FD	OPS_CVTSP	= 00000009
OPS_ADDH2	= 000060FD	OPS_CVTTP	= 00000026
OPS_ADDH3	= 000061FD	OPS_CVTWD	= 0000006D
OPS_ADDP4	= 00000020	OPS_CVTWF	= 0000004D
OPS_ADDP6	= 00000021	OPS_CVTWG	= 00004DFD
OPS_ASHP	= 000000F8	OPS_CVTWH	= 00006DFD
OPS_CLRD	= 0000007C	OPS_DIVD2	= 00000066
OPS_CLRF	= 000000D4	OPS_DIVD3	= 00000067
OPS_CLRG	= 0000007C	OPS_DIVF2	= 00000046
OPS_CLRH	= 00007CFD	OPS_DIVF3	= 00000047
OPS_CMPC3	= 00000029	OPS_DIVG2	= 000046FD
OPS_CMPC5	= 0000002D	OPS_DIVG3	= 000047FD
OPS_CMPD	= 00000071	OPS_DIVH2	= 000066FD
OPS_CMPP	= 00000051	OPS_DIVH3	= 000067FD
OPS_CMPPG	= 000051FD	OPS_DIVP	= 00000027
OPS_CMPPH	= 000071FD	OPS_EDITPC	= 00000038
OPS_CMPP3	= 00000035	OPS_EMODD	= 00000074
OPS_CMPP4	= 00000037	OPS_EMODF	= 00000054
OPS_CRC	= 00000008	OPS_EMODG	= 000054FD
OPS_CVTBD	= 0000006C	OPS_EMODH	= 000074FD
OPS_CVTBF	= 0000004C	OPS_LOCC	= 0000003A
OPS_CVTBG	= 00004CFD	OPS_MATCHC	= 00000039
OPS_CVTBH	= 00006CFD	OPS_MNEGD	= 00000072
OPS_CVTDB	= 00000068	OPS_MNEGFI	= 00000052
OPS_CVTDF	= 00000076	OPS_MNEGG	= 000052FD
OPS_CVTDH	= 000032FD	OPS_MNEGH	= 000072FD
OPS_CVTDL	= 0000006A	OPS_MOVD	= 00000070
OPS_CVTDW	= 00000069	OPS_MOVF	= 00000050
OPS_CVTFB	= 00000048	OPS_MOVG	= 000050FD
OPS_CVTFD	= 00000056	OPS_MOVH	= 000070FD
OPS_CVTFG	= 000099FD	OPS_MOVP	= 00000034
OPS_CVTFH	= 000098FD	OPS_MOVTC	= 0000002E
OPS_CVTFL	= 0000004A	OPS_MOVTUC	= 0000002F
OPS_CVTFW	= 00000049	OPS_MULD2	= 00000064
OPS_CVTGB	= 000048FD	OPS_MULD3	= 00000065

OPS_MULF2 = 00000044
 OPS_MULF3 = 00000045
 OPS_MULG2 = 000044FD
 OPS_MULG3 = 000045FD
 OPS_MULH2 = 000064FD
 OPS_MULH3 = C00065FD
 OPS_MULP = 00000025
 OPS_POLYD = 00000075
 OPS_POLYF = 00000055
 OPS_POLYG = 000055FD
 OPS_POLYH = 000075FD
 OPS_SCANC = 0000002A
 OPS_SKPC = 0000003B
 OPS_SPANC = 0000002B
 OPS_SUBD2 = 00000062
 OPS_SUBD3 = 00000063
 OPS_SUBF2 = 00000042
 OPS_SUBF3 = 00000043
 OPS_SUBG2 = 000042FD
 OPS_SUBG3 = 000043FD
 OPS_SUBH2 = 000062FD
 OPS_SUBH3 = 000063FD
 OPS_SUBP4 = 00000022
 OPS_SUBP6 = 00000023
 OPS_TSTD = 00000073
 OPS_TSTF = 00000053
 OPS_TSTG = 000053FD
 OPS_TSTH = 000073FD
 OPCODE = 00000000
 OPCODE_BASE = 00000029
 OPCODE_MAX = 0000003B
 OPERAND_1 = 00000008
 OPERAND_2 = 0000000C
 OPERAND_3 = 00000010
 OPERAND_4 = 00000014
 OPERAND_5 = 00000018
 ***** X 02
 ***** X 02
 00000000 RG 02
 00000034 RG 02
 00000080 RG 02
 ***** X 02
 ***** X 00
 ***** X 00

+-----+
 ! Psect synopsis !
 +-----+

PSECT name

	Allocation	PSECT No.	Attributes																
ABS	00000000 (0.)	00 (0.)	NOPIC	USR	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE						
\$ABSS	00000000 (0.)	01 (1.)	NOPIC	USR	CON	ABS	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE						
_VAX\$CODE	0000008C (140.)	02 (2.)	PIC	USR	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	QUAD						

```
+-----+  
! Performance indicators !  
+-----+
```

Phase	Page faults	CPU Time	Elapsed Time
Initialization	9	00:00:00.05	00:00:01.44
Command processing	71	00:00:00.79	00:00:04.49
Pass 1	409	00:00:13.70	00:00:39.56
Symbol table sort	0	00:00:00.65	00:00:01.11
Pass 2	131	00:00:05.51	00:00:21.29
Symbol table output	19	00:00:00.15	00:00:00.37
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	641	00:00:20.87	00:01:08.29

The working set limit was 1500 pages.

82473 bytes (162 pages) of virtual memory were used to buffer the intermediate code.
There were 30 pages of symbol table space allocated to hold 453 non-local and 1 local symbols.
4755 source lines were read in Pass 1, producing 13 object records in Pass 2.
141 pages of virtual memory were used to define 139 macros.

```
+-----+  
! Macro library statistics  
+-----+
```

Macro library name	Macros defined
\$255\$DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1	3
\$255\$DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	8

518 GETS were required to define 8 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LISS:BOOSEMULATE/OBJ=OBJ\$:BOOSEMULATE MSRC\$:BOOTSWT/UPDATE=(ENH\$:BOOTSWT)+MSRC\$:MISSING/UPDATE=(ENH\$:MISSING)+MSRC\$:VAXEMULAT/

0142 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

